Mobisaic

An Information System for a Mobile Wireless Computing Environment

Geoffrey M. Voelker and Brian N. Bershad

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Technical Report 95-04-01

Abstract

Mobisaic is a World Wide Web information system designed to serve users in a mobile wireless computing environment. Mobisaic extends the Web by allowing documents to both refer and react to potentially changing contextual information, such as one's current location in the wireless network. Mobisaic relies on client-side processing of HyperText Markup Language documents that support two new concepts: *Dynamic Uniform Resource Locators (URLs)* and *Active Documents.* A dynamic URL is one whose results depend upon the state of the user's mobile context at the time it is resolved. An active document automatically updates its contents in response to changes in a user's mobile context. This paper describes the design of Mobisaic, the mechanism it uses for representing a user's mobile context, and the extensions made to the syntax and function of Uniform Resource Locators and HyperText Markup Language documents to support mobility.

Mobisaic: An Information System for a Mobile Wireless Computing Environment

Geoffrey M. Voelker and Brian N. Bershad
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

Abstract

Mobisaic is a World Wide Web information system designed to serve users in a mobile wireless computing environment. Mobisaic extends the Web by allowing documents to both refer and react to potentially changing contextual information, such as one's current location in the wireless network. Mobisaic relies on client-side processing of HyperText Markup Language documents that support two new concepts: *Dynamic Uniform Resource Locators (URLs)* and *Active Documents*. A dynamic URL is one whose results depend upon the state of the user's mobile context at the time it is resolved. An active document automatically updates its contents in response to changes in a user's mobile context. This paper describes the design of Mobisaic, the mechanism it uses for representing a user's mobile context, and the extensions made to the syntax and function of Uniform Resource Locators and HyperText Markup Language documents to support mobility.

1 Introduction

This paper describes Mobisaic, a system that uses the World Wide Web [**?**] to enable information browsing in a mobile computing environment. Information browsing is an ideal mobile application because it allows users to interact with their environment as they work within it; it places minimal requirements on a user-input device; and it cannot be handled with large on-board caching. With mobile information browsing, users can discover who and what is in their immediate surroundings, whether it is colleagues at a business meeting, speakers at a presentation, projects in a lab, or displays in a museum. A mobile information system can also allow users to execute general queries that incorporate information from their environment, such as finding the nearest cafe or the nearest bus stop that will take them to a specific location.

Users of the World Wide Web (WWW) rely on client browsers to access information servers on the Internet. With Web clients, users browse documents written in the HyperText Markup Language (HTML) by traversing hypertext links, called Uniform Resource Locators (URLs), that load documents or invoke programs on the information servers.

Mobisaic extends standard client browsers to take advantage of mobility in two ways. First, Mobisaic allows authors to reference dynamic information, such as a user's location, in hypertext links called *dynamic URLs*. When the user traverses a dynamic URL, the client resolves any references to dynamic information it may contain and sends the result to the server. Second, Mobisaic supports *active documents*, documents that present and automatically update information for the user as the information they contain changes or otherwise becomes invalid. The update is done by the client browser, which receives notifications when the dynamic information changes.

Dynamic information in Mobisaic is represented using dynamic environments [**?**]. Just as standard UNIX shells provide environment variables to customize applications started from the shell (e.g., **DISPLAY**), dynamic environments provide environment variables for customizing mobile applications. For example, a dynamic document might include a reference to the **Location** dynamic environment variable to customize its contents according to the user's current location.

1.1 Related work

Researchers at Xerox PARC have broadly introduced the idea of context-aware applications [?]. They also initially proposed the notion of dynamic environments as a means of representing and disseminating information from users' mobile contexts throughout the system. Mobisaic is essentially an application of those ideas to WWW browsing.

Joel Bartlett at DEC WRL has investigated the feasibility of implementing and using a Web client on a handheld personal digital assistant (PDA) in the Wireless World Wide Web (W4) project [?]. The W4 system consists of a Web client that runs on an Apple Newton MessagePad and communicates with the wired network via a ~2400 baud Motorola CELLect modem and MicroTac phone. The Web client is structured so that a workstation host on the remote end of the modem link does most of the document formatting, thereby relieving the less powerful PDA of the CPU intensive formatting task and conserving as much of the limited wireless bandwidth as possible. The W4 project has shown that current PDAs are feasible platforms for running Web clients when care is taken in implementing the client.

The Dynamic Documents project at MIT has also investigated providing mobile access to the World Wide Web [?]. Dynamic Documents are Tcl scripts that, once received, are executed in a "safe" Tcl interpreter in a modified Mosaic Web client. The output of the Tcl script is an HTML document that is displayed by the client. Dynamic Documents can interact with the Web client to, for example, configure their output for displaying on small displays. In addition, Dynamic Documents can use the Tk library to create their own user interfaces to implement more complex Web applications from within the client.

The primary difference between the Dynamic Documents project and Mobisaic is that the Dynamic Documents project has focused on the problems of overcoming the limited bandwidth of the wireless communication link and the display limitations of mobile devices, whereas Mobisaic has focused on enhancing the usefulness of the Web by incorporating information from the user's mobile computing context into the system.

1.2 Paper outline

The rest of this paper is organized as follows. Section 2 gives an overview of the World Wide Web system, and describes the extensions to the system used for incorporating a user's mobile computing context into the WWW. Sections 3 and 4 describe dynamic URLs and active documents. Section 5 discusses how Mobisaic can be useful in the desktop environment as well as the mobile environment. Section 6 describes the implementation of Mobisaic. Section 7 discusses future work, and section 8 summarizes. Two appendicies are also included for reference. The first details the implementation of dynamic environments, and the second describes the Mobisaic client library interface.

2 System overview

This section first provides a high-level overview of the World Wide Web information system, and then describes the extensions used for incorporating a user's mobile computing environment into the system.

2.1 The World Wide Web

The three main components of a World Wide Web (WWW) information system are documents, clients, and information servers. The user interacts with the system using a Web client, which lets the user name and load documents from servers for viewing. Web clients typically support a number of different document types, such as files available via ftp, netnews, and Hypertext Markup Language (HTML), and support connections with a variety of information servers, such as ftp daemons, news servers, and Hypertext Transport Protocol (HTTP) daemons. Documents are files on a server referenced by Uniform Resource Locators (URLs), and they can contain a variety of information types, including ASCII text formatted according to HTML directives, embedded pictures, and audio and video clips, as well as embedded URLs that are used as hypertext links to other documents. URLs can also name programs on HTTP daemons that, when executed, produce an HTML document as output.

2.2 Extensions for a mobile WWW

In its current form, the Web infrastructure cannot easily accommodate mobile clients because the dynamic information it supports is either returned from the server without incorporating any user context at all, or is incorporated explicitly using forms-based interfaces that require user input on the client. Moreover, there is no support for automatically updating a document when it, or the reason for displaying it, changes.

To better support the use of dynamic information, we have extended the Web infrastructure to include:

- A network server that maintains mobile computing contexts within a client-specific domain;
- An asynchronous callback mechanism to notify Web clients when a user's dynamic computing environment changes;
- A syntax for referencing dynamic information in URLs and documents.

Representing mobile computing contexts.

Mobisaic uses dynamic environments to represent a user's mobile computing context. The basic unit in a dynamic environment is the dynamic environment variable, which is conceptually similar to a standard UNIX shell environment variable; dynamic environment variables have a name and a value, and they can be accessed and changed to customize applications to the user's mobile computing environment just as shell environment variables customize applications launched from the shell. However, unlike shell environment variables that are associated with a login process, dynamic environment variables are associated with users and places, and have indefinite lifetimes. Applications on the network with sufficient privilege can access and change dynamic environment variables, and whatever changes they make can be seen by other applications with sufficient access privileges.

Notification of changes in mobile computing contexts.

Active documents allow environmental changes to be reflected in the information displayed to the user. If the information in an active document becomes invalid, then the client can be notified of the change so that it can display a more relevant document. Included in the notifications are the name of the variable that changed and its new value.

For example, say that the user changes cell locations in the wireless environment. The wireless communications system that is monitoring the user's location can publish the new location by updating the **Location** variable in the user's dynamic environment. If the user were displaying a document that was sensitive to location, the client would have subscribed to the **Location** variable, and would receive a notification informing it of the change. At this point, the client could take action in response to the change, such as loading a new document that relates to the user's new location.

Syntax and scope.

A Mobisaic client relies on a syntax for referencing dynamic environment variables within dynamic URLs and active documents. The syntax supported by Mobisaic is of the form **$(***environment.variable***)**, where *environment* and *variable* denote a dynamic environment and dynamic environment variable, respectively. For example, **$(voelker.Location)** would reference the name of my current location in the wireless network. Mobisaic also supports a shorthand notation for referencing variables in the user's own dynamic environment. If the reference doesn't contain the name of an environment, then Mobisaic assumes that the variable referenced is in the environment associated with the user. Thus, **$(Location)** refers to the **Location** environment variable in the user's dynamic environment.

Mobisaic supports recursive references to dynamic environment variables so that environment or variable names can themselves be references to dynamic environment variables. For example, given that locations have dynamic environments associated with them, **\$(\$(Location).Printer)** would reference the **Printer** variable in the environment associated with the user's current location.

3 Using dynamic URLs

Dynamic URLs allow a single URL to return different documents or execute different commands depending upon the state of the user's dynamic environment at the time the URL is selected. A URL is dynamic if it references at least one dynamic environment variable. For example, in our department we have written HTML documents describing ourselves and the places in which we work (see Figure 1). The name space of these documents on our server is well structured, enabling the following dynamic URL to return the document describing the user's current location:

http://www/places/\$(Location).html [1]

Another example of a dynamic URL is a Web server that has a program **busroute** which takes a starting location, a destination, and a time as arguments, and returns an HTML document detailing how to get to the closest bus stop on the shortest bus route to the destination. A dynamic URL to find the bus route to the Space Needle in Seattle would appear as:

http://www/htbin-post/voelker/busroute?\$(Location)+SpaceNeedle+\$(Time.TIME)

Note that the question mark and plus signs in the query are standard HTML syntax denoting the arguments that are passed to the program invoked on the server.

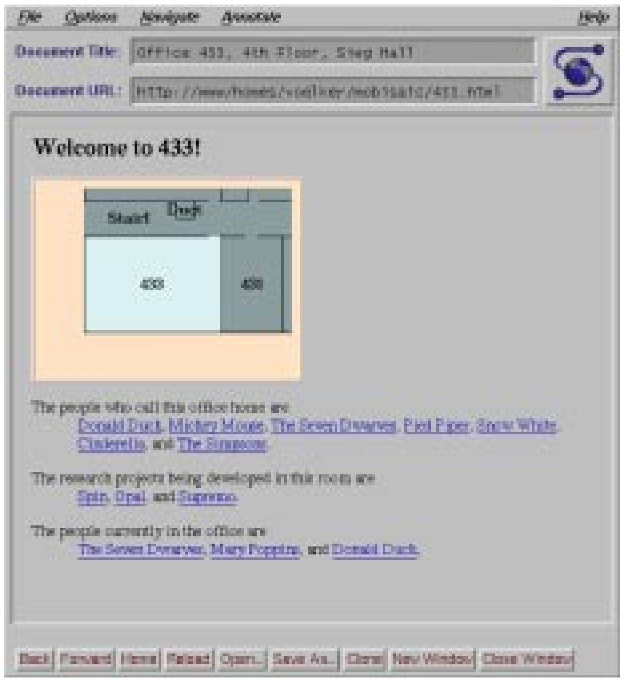
3.1 Resolving dynamic URLs

When a user selects a dynamic URL in a document, the client browser is responsible for resolving all references to dynamic environment variables within the URL. The client obtains the values of dynamic environment variables from the appropriate dynamic environment and replaces the references with the values as strings. When all variable references have been resolved, the result is a standard URL that the client then sends to the server. For example, if a user were in office 433 and selected the location description dynamic URL in the previous section, the client would resolve the **Location** dynamic environment variable in the user's context and send the following URL to the server:

http://www/places/433.html

Having the Mobisaic client resolve the dynamic environment variable references gives the most flexibility to the system. The variable references could have been resolved in two other places: the application, if the dynamic URL named a program on the server to execute, or the server. If applications had to resolve the variable references, then dynamic URLs would be limited to naming only those applications that were modified to understand and use dynamic environment variables. Likewise, if the server were to resolve the references, then dynamic URLs would be limited to using only dynamic environment aware servers. When the client resolves the references, however, dynamic URLs can evaluate to a URL that names any document or application on any server that a standard URL can name.

[1]For brevity, we use *www* in place of our server at *www.cs.washington.edu.*



Office 433, 4th Floor, Sieg Hall

http://www/homes/voelker/mobisaic/433.html

Welcome to 433!

The people who call this office home are
Donald Duck, Mickey Mouse, The Seven Dwarves, Pied Piper, Snow White, Cinderella, and The Simpsons.

The research projects being developed in this room are
Spin, Opal, and Supremo.

The people currently in the office are
The Seven Dwarves, Mary Poppins, and Donald Duck.

Figure 1: The WhereAmI active document, referenced by http://www/places/$(Location).html

4 Active documents

Active documents are HyperText Markup Language documents that enable the Web client to automatically react to changes in a user's mobile computing context. They give the client the ability to update the information being displayed without the user having to navigate the Web, and place less of a burden on the user to search for information by placing more of a burden on the author to organize it. This tradeoff is possible in a mobile environment because users who roam the wireless network are quite likely to be interested about who and what is in their immediate surroundings, and the information in a user's mobile context is enough to enable a Web client to do the searching on the user's behalf. In this way, it changes the way users interact with the Web: they spend less time searching for information because the client presents it to them as they interact with their surroundings.

This section describes how to write active documents and how the client handles them when they are being viewed by the user. To illustrate these processes, it also describes a set of pages collectively called the WhereAmI active document, which is a guide for visitors to our department. Each page in it corresponds to a place in our wireless network, and contains a brief description of the place, links to the home pages describing the occupants of the place, and links to pages describing any projects housed in the place. When a user selects this active document, the client loads the page corresponding to the user's location. Then, as the user changes rooms, the client automatically discards the page describing the old room and replaces it with the one describing the new room. Each page in the WhereAmI active document can be loaded using the dynamic URL from the previous section:

http://www/places/$(Location).html

Authors write active documents just like they write standard HTML documents, with one addition. They must place a *subscribe* command in the document which lists the dynamic environment variables that the client must subscribe to when it loads the document. In effect, the variables listed in the subscribe command are the elements of a user's mobile context that, when they change value, invalidate the information in the document. The new values of the variables also provide the information necessary for the client to determine which document to load in place of the current one.

A subscribe command is embedded in an HTML comment line. The command has the following form:

<!– (**subscribe to** *variable variable ...*) –>

Variable is a standard reference to a dynamic environment variable. When the client loads the document and parses subscribe commands, it subscribes to each variable specified in the command.

When the client receives a notification for a subscribed variable indicating that the variable has changed value, the new value of the variable in the notification determines what action the client will take in the face of this notification. The new value of the variable can be an explicit directive to the client:

reload Re-execute the URL that loaded the current document. If the URL is dynamic, the references to dynamic environment variables are resolved again.

load URL Execute a new URL and load the document in the same window.

spawn URL Execute a new URL and load the document in a new window.

close Close the current window.

Otherwise, the client does not interpret the new value and simply reloads the document. If the URL naming the document is a dynamic URL, then the client will evaluate the dynamic URL as if the user had selected it.

For the WhereAmI active document, each page has the following subscribe command:

<!– (**subscribe to** $(Location)) –>

The command tells the client that, when it loads the page, it should subscribe to the **Location** variable in the user's dynamic environment. When this variable changes value, the client will receive a notification with the new value of **Location**. Since the value is not an explicit command to the client, it will ignore the value itself and use the notification as a signal to reload the active document. And since the URL naming the document is dynamic, the client will re-evaluate it and load the active document that describes the user's new location.

Arbitrary client notifications

In addition to receiving notifications for any active documents the client is displaying, the client can also receive notifications that do not refer to any of its displayed documents. To receive such notifications, the client subscribes to the **MOBISAIC-STREAM** variable in the user's dynamic environment. Any process can then send notifications to the client to, for example, spawn a window and load a new document that might be of interest to the user.

5 Mobisaic on the desktop

Although Mobisaic was originally inspired for use in a mobile computing environment, it can also be useful in the desktop environment. There are a number of information sources in the WWW that produce information periodically, and it is straightforward to write documents in Mobisaic that tap into these sources. Some documents that have already been written include simple daily scripts that, early in the morning, publish the URLs for the Dr. Fun and Dilbert comic pages to a list of interested users running Mobisaic. The published URLs spawn new windows showing the contents of the pages. When users come in to work in the morning, the daily comic pages are already showing on their screens.

As another example, we use active documents together with electronic mail to implement a distributed, recommendation-based "hot list" of new and interesting pages. Our local departmental version of Mosaic has been modified to make it easy to forward URLs to others in our department using email. It is now common practice for people in the department to forward to friends URLs that they have discovered and find interesting. The email messages generated by Mosaic have a special mail tag, *X-URL*, that contains the URL being forwarded. A user's incoming mail filter detects these tags and publishes the URLs in them to the user's Web client. The result is that, when users have a URL forwarded to them, a window automatically appears on their screen displaying the document referenced by the URL.

A third application uses active documents to display and update stock quotes. A background filter monitors a stock quote information source, and, in a dynamic environment for stock prices, publishes the latest values of the stocks it monitors as they change. Documents displaying the information for a given stock subscribe to the dynamic environment variable associated with the stock in the stock dynamic environment. When users view a document describing or referencing a stock, the document will update itself as new stock values are published in the dynamic environment variable for stocks.

6 Implementation

This section discusses our implementation of Mobisaic and the changes we applied to a standard Web client to use active documents and dynamic URLs.

6.1 Dynamic environments

Dynamic environments [?] support a network-based publish and subscribe paradigm [?]. A dynamic environment is maintained by a subscription-based server to which applications broadcast queries and from which applications receive multicast notifications. We use the Zephyr notification system [?] to implement the publish and subscribe facilities. For detailed information about how Zephyr message classes and subscriptions were used to implement the communication aspects of dynamic environments, please see appendix A at the end of the paper.

We chose to use Zephyr because it provides network transparency, automatic subscriber-based routing, authentication, asynchronous notification, and the ability to run redundant Zephyr clients supporting redundant instances of the same dynamic environment. (We presently do not take advantage of Zephyr's authentication system or support for redundancy). Zephyr is not without its drawbacks, however. It operates only within relatively small administrative domains, such as a department or campus. It cannot distribute information quickly to many hosts, which can become a problem in the current system during heavy load. (This bottleneck and a possible solution to it are discussed in [?].) Fortunately, a C library interface hides the use of Zephyr from Web clients, so changing to a new transport should be relatively easy.

6.2 Client modifications

Any Web browser can be modified to support dynamic URLs and active documents provided that it supports an interface for loading and reloading documents, spawning and closing windows, and the ability to add an asynchronous input descriptor to its set of inputs. A client library handles all communication with dynamic environments, and parses dynamic environment variable references (see appendix B for a detailed description of the client library interface). Filters, supplied by the library, are applied to the input and output communication paths. The filter on the output stream resolves references to dynamic environment variables embedded in dynamic queries, and the filter on the input stream subscribes the client to dynamic environment variables embedded in active documents. If the client supports an internal interface for document and window manipulation, then it can be directly linked with the Mobisaic client library. For those clients that only support an external interface or do not have an interface for adding asynchronous input descriptors, we have a wrapper program that handles dynamic environments and controls the Web client as a child process.

Our prototype Mobisaic client is the X Mosaic client [?] extended with the Mobisaic client library. The X Mosaic client has a relatively clean internal interface for loading documents and spawning windows, so most of the code changes were to add the file descriptor for the dynamic environment communication channel to the list of input descriptors, the callback to handle asynchronous input on the descriptor, and calls to the Mobisaic library filters on the input and output communication paths. Overall, our changes added less than 60 lines to the client.

7 Future Work

This section discusses some of the limitations of the current design and implementation of Mobisaic, and suggests possible approaches for overcoming these limitations.

7.1 Callbacks

Integrating the subscribe mechanism of dynamic environments into Web clients effectively provides a callback mechanism for other applications to notify the client that the state of the document the client is viewing has changed in some respect. However, it is not a general callback mechanism for the World Wide Web since it requires the use of dynamic environments. The Internet Engineering Task Force (IEFT) is in the process of designing a standard callback mechanism with which Web servers and other applications can communicate asynchronously with Web clients, and Mobisaic's use of dynamic environments should be compatible with this callback standard once it emerges.

7.2 Storing URLs

Uniform Resource Locators are stored in various collections, such as hotlists, session and global histories, indices, bookmarks, and even email, to aid the user in organizing access to interesting documents. Storing dynamic URLs raises an interesting issue about what actually is stored because the user or Web client now has the choice of storing either the dynamic URL itself, or the result of evaluating the URL at the time it is stored.

One can imagine wanting to do both in different situations. For instance, assume a user has selected the first location-based dynamic URL from section 3 and finds the loaded document interesting. If the user wanted to store this document in a hotlist, the user would want to store the evaluated dynamic URL. However, if the user wanted to store a reference to the WhereAmI document, the user would want to store the dynamic URL itself in the hotlist so that, the next time it is selected, the dynamic URL would be evaluated in the user's current context.

To address this problem, Mobisaic could store evaluated dynamic URLs by default, and allow users to explicitly override the default behavior with a menu option. However, such behavior would require more extensive changes to each Web client supporting the Mobisaic extensions since these changes need to be made to the user interface.

7.3 Disabling active documents

Active documents in Mobisaic are always active in the sense that they will react to notifications to their subscriptions whenever they receive them. This behavior may not always be desirable. For example, imagine that you are in a museum that used active documents to describe its exhibits and you have just walked up to a particularly interesting exhibit. It is so interesting, in fact, that you want to show you friend just down the hall the document that has appeared in your browser. If you walk down the hall to show your friend the document, though, it may no longer be in the browser, having since been replaced with the document that describes the exhibit in front of your friend.

Mobisaic should provide some mechanism for allowing the user to disable active documents, such as with a button at the bottom of the document window, although, again, this would require more client-specific changes.

7.4 Subscriptions

The current method of subscribing active documents to dynamic environment variables is inflexible in two ways. The ability to express interest in combinations of multiple dynamic environment variables is limited. Furthermore, users might want subscriptions in some active documents to continue to hold even after the document is no longer being viewed.

Variable expressions in subscriptions

Subscriptions to multiple dynamic environment variables implicitly form an *OR* clause of those variables since the client subscribes to all variables and receives notifications when any of them change. A more expressive syntax would allow authors of active documents to subscribe their documents to arbitrary logical expressions of dynamic environment variables and arithmetic and string operators.

Our implementation of dynamic environments could be extended to support such subscriptions so that, whenever any of the dynamic environment variables referenced in the expression change, the expression included in the subscription would be evaluated. If the expression evaluates to a non-zero value, the client would then be notified of the new value of the variable that changed and, possibly, the value of the evaluated expression. With such a syntax, an author might subscribe a location-based active document using the following expression that disables the active document in, for example, the restroom:

<!– (**subscribe to** $(Location) && ($(Location) != "Restroom")) –>

As another example, the following subscription might be used to receive updates when the stock **AStock** falls below a certain threshold. The stock price is maintained in the dynamic environment **Stocks**, and the threshold **AStockThreshold** is maintained in the user's dynamic environment:

<!– (**subscribe to** $(Stocks.AStock) && ($(Stocks.AStock) < $(AStockThreshold))) –>

Indefinite subscriptions

One feature of Mobisaic is that active document subscriptions only last as long as the document is being browsed. However, this feature is also a limitation. Consider the situation where a user is exploring a building while viewing the WhereAmI active document, using it as a starting point to explore other documents when entering a new room. Once the user loads another document instead of the WhereAmI active document, though, the location-based subscription is lost. Consequently, when the user moves to a different place, no notifications are sent to the client even though the user might still be interested in receiving them.

It is possible to address this problem with the current version of Mobisaic by having whatever process that updates the **Location** dynamic environment variable also send a **goto** command to the user's Web client subscription, **MOBISAIC-STREAM**, the variable to which the client is always subscribed. However, this approach to the problem is messy in the sense that whatever is updating **Location** now has to know about the Mobisaic application.

Another possibility is to have a button or command in the Web client toggle the scope of subscriptions between single documents or all documents, although this would seem to require too much user interaction. Alternatively, Mobisaic could support a **(subscribe indefinitely to)** syntax that would give active documents the ability to have the client make document-specific subscriptions, such as to **$(Location) != "Restroom"**, and indefinite subscriptions, such as to **$(Location)**. A button or command could then be provided by the Web client to clear all indefinite subscriptions when the user finished browsing, for example, the set of documents comprising the WhereAmI application.

8 Summary

This paper has described a World Wide Web information system called Mobisaic that investigates information browsing in a mobile wireless computing environment. Mobisaic introduces two mechanisms, dynamic URLs and active documents, for incorporating contextual information from a user's mobile computing environment into the Web. Dynamic URLs allow a single URL to return different documents or execute different commands depending upon the values of the embedded variables at the time the URL is selected by the user. Active documents specify dynamic environment variables to which client subscribe. With these subscriptions, clients respond to changes in the user's mobile context and update the active documents being browsed.

Minimal modifications are required to Web clients and the URL syntax to support the features of Mobisaic. Web servers need no modifications whatsoever. A library hides the details of the communication mechanism and provides filters for parsing and resolving dynamic environment variable references, making it straightforward to modify a Web client to take advantage of the features of the Mobisaic Web system.

The X Mosaic Web client has been modified to use the Mobisaic extensions, and is currenty running on PC laptops running the Linux operating system. The laptops are mobile and communicate with the World Wide Web using Proxim RangeLan2 wireless ethernet PCMCIA cards. The WhereAmI application described in this paper is accessible via the first dynamic URL from section 3. An infrared receiver attached to each laptop's serial line detects transmissions from infrared beacons placed in rooms and hallways, providing the fine-grained location information needed to make the WhereAmI application useful. Work is ongoing on incorporating the Mobisaic extensions with a second Web client, the W* client running in the Wit [**?**] mobile environment.

Acknowledgements

I would like to thank Brian Bershad, Marc Fiuczynski, Ed Lazowska, Hank Levy, Stefan Savage, Terri Watson, George Forman, and John Zahorjan for their suggestions on Mobisaic. Bill Schilit at Xerox Parc has been incredibly helpful in discussing context-aware applications. Finally, special thanks are due to Xerox PARC for supplying us with the software and hardware that began our experiment in mobile computing.

References

[Andreessen and Bina 94] Marc Andreessen and Eric Bina. "NCSA Mosaic: A Global Hypermedia System" In *Internet Research*, 4(1):7–17, Spring 1994.

[Bartlett 94] Joel F. Bartlett. "W4 — the Wireless World Wide Web." In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, December, 1994.

[Berners-Lee et al. 92] Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bernard Pollermann. "World-Wide Web: The Information Universe" In *Electronic Networking: Research, Applications, and Policy*, 2(1): 52–58, Spring 1992.

[DellaFera et al. 88] C. Anthony DellaFera, Mark W. Eichen, Robert S. French, David C. Jedinsky, John T. Kohl, and William E. Sommerfeld. "The Zephyr Notification Service." In *Proceedings of the USENIX 1988 Winter Conference*, Winter 1988.

[Oki et al. 93] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. "The Information Bus — An Architecture For Extensible Distributed Systems" In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, December 1993.

[Kaashoek et al. 94] M. Frans Kaashoek, Tom Pinckney, and Joshua A. Tauber. "Dynamic Documents: Mobile Wireless Access to the WWW." In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, December, 1994.

[Schilit and Theimer 94] Bill N. Schilit and Marvin M. Theimer. "Disseminating Active Map Information to Mobile Hosts" *IEEE Network*, September, 1994.

[Schilit et al. 93A] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. "The ParcTab mobile computing system." In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pp. 34–39, October 1993.

[Schilit et al. 93B] Bill N. Schilit, Marvin Theimer, and Brent B. Welch. "Customizing mobile applications." In *Proceedings of the USENIX Symposium on Mobile & Location-Independent Computing*, pp. 129–139, August 1993.

[Watson 94] Terri Watson. "Application Design for Wireless Computing." In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, December, 1994.

A Zenv: Dynamic Environments using Zephyr

Zenv is an implementation of dynamic environments using the Zephyr notification system. This appendix gives a brief introduction to the Zephyr system, and then details how Zenv uses Zephyr to implement dynamic environment servers and clients.

A.1 Introduction

Zephyr is a notification system designed to support lightweight network communication (as compared to email) among groups of people using different machines connected via a network. Zephyr emphasizes sending messages that are simple to address and compose, and that can be quickly consumed by recipients.

Users can subscribe themselves to message classes, class instances of those classes, and recipients. By convention, communication among people is done using the message class **message**. Class instances of the message class name a topic of interest, such as **lunch** for people who like to meet for lunch, or **C109D** for people who share an office. A user who wanted to receive lunch messages would therefore subscribe to **(message, lunch, *)**, indicating that the user was interested in receiving messages sent to the message class **message**, the class instance **lunch**, and the recipient *****, which denotes the special recipient "everyone". A user who wanted to send a message to the lunch group would send a message that matched the lunch subscription, except that no recipient would be specified.

The following sections describe how Zenv uses these subscription and message facilities to implement the dynamic environment servers and clients.

A.2 Dynamic environment servers

A dynamic environment is maintained by a dynamic environment server, **zenv**, which is a program that subscribes itself to the Zephyr system and processes requests from clients of dynamic environments. To maintain the **voelker** dynamic environment, for example, a dynamic environment server would subscribe to the message class **ENV_SERVER**, class instance **voelker**, and recipient *****.

Requests to a dynamic environment server are placed in the message text itself. The servers currently support the following commands in messages:

`zgetenv` *variable*
: Get the value of variable.

`zsetenv` *variable* `=` *value*
: Set the value of a variable, creating the variable if necessary.

`zunsetenv` *variable*
: Remove the variable from the dynamic environment.

`zprintenv`
: Get the values of all variables in the dynamic environment.

`zcloseenv`
: Close the dynamic environment.

When the server receives `zgetenv`, `zsetenv`, and `zprintenv` commands, it sends a response to the sender. The response is sent to the Zephyr subscription **(ENV_CLIENT,** ***environment*****,** ***sender*****)**, and the response itself is placed in the text of the message. By specifying the sender as the recipient of the message, the server ensures that only the sender will receive the response.

When the server receives a `zsetenv` request to update the value of a variable, it sends out an update message to all dynamic environment clients who have subscribed to that variable. To accomplish this, the server sends a message to the Zephyr subscription (**ENV_CLIENT,** *environment*, *variable*), and places the new value of the variable in the text of the message. For example, if the **Location** variable were updated to the value **C109D** in the **voelker** dynamic environment, then the server maintaining the environment would send a message to **(ENV_CLIENT, voelker, Location)** with **C109D** as the text. The Zephyr system itself takes care of delivering the message to all recipients who have subscribed to this update subscription.

A.3 Dynamic environment clients

Clients of dynamic environments can make requests to dynamic environment servers by sending messages to the servers using the message specification described in the previous section. For example, to set the value of the variable **Location** in the dynamic environment **voelker**, a client would send a message to the subscription **(ENV_SERVER, voelker, *)** and place the command `zgetenv Location` in the text of the message.

To receive update notifications of variables in dynamic environments, clients subscribe to the Zephyr subscription described in the previous section, (**ENV_CLIENT,** *environment*, *variable*). Whenever a variable that the client has subscribed to changes, the server sends a message to this subscription and the client receives it.

A.3.1 Zephyr programs as clients

Since Zenv is built using Zephyr, the standard zephyr programs can be used to send commands to dynamic environment servers. For example, zwrite can be used to set the value of the **Location** variable in the **voelker** dynamic environment:

```
% zwrite -c ENV_SERVER -i voelker
Type your message now. End with control-D or a dot on a line by itself.
zsetenv Location = C109D
.
```

Any of the dynamic environment commands (zgetenv, zsetenv, zunsetenv, and zprintenv) may be used in the text of a zwrite message to a dynamic environment server.

To receive responses and updates from a dynamic environment server using the **zwgc** program (the program normally used to receive and display messages from other people), users need to place subscriptions in their `$HOME/.zephyr.subs` file. The following is an example list of such subscriptions:

```
ENV_CLIENT,voelker,*
ENV_CLIENT,voelker,Location
ENV_CLIENT,voelker,
```

The first subscription tells **zwgc** to report all updates to any variables in the **voelker** dynamic environment. The second subscription tells **zwgc** to report all updates to the **Location** variable in the **voelker** dynamic environment. And the third subscription tells **zwgc** to report responses to commands the user sends to the **voelker** dynamic environment.

A.3.2 Zenv client programs

The Zenv system also provides a set of client programs for accessing and manipulating dynamic environments. The following programs are provided:

`zgetenv [ -env` *environment* `] [ -f ] [ -h ]` *var*

Zgetenv returns the value of a variable in a dynamic environment. When used with the -f option, it will indefinitely print out updates to a variable by waiting for the variable to change value.

zsetenv [-env *environment*] [-h] *var* [=] *value*

Zsetenv sets the value of a variable in a dynamic environment.

zunsetenv [-env *environment*] [-h] *var*

Zunsetenv removes variables from dynamic environments.

zprintenv [-env *environment*] [-f] [-h]

Zprintenv prints the values of all variables in a dynamic environment. When used with the -f option, it will indefinitely print out updates to any variable in the environment by waiting for variables to change value.

zcloseenv [-env *environment*] [-h]

Zcloseenv closes a dynamic environment.

A.3.3 libzenvclient.a

The Zenv client library provides an interface for writing clients that use the Zenv dynamic environment system. It exports the following sets of types and routines:

```
/*
 * Callback structure used in ZenvSubscribeLoop and ZenvSubscribeToVarLoop.
 */
typedef int (*ZEnvCallback)(void *, char *, char *);
typedef struct {
    ZEnvCallback callback;
    void        *data;
} ZEnvClosure;

/*
 * Return values from the library routines.  Error values are always negative.
 */
#define ZENV_SUCCESS                     0
#define ZENV_UNKNOWN_ENVIRONMENT        -1
#define ZENV_UNKNOWN_VARIABLE           -2
#define ZENV_INVALID_ARG                -3

/*
 * Initialize the ZEnv client library.
 */
extern void ZEnvInitialize ();

/*
 * Return the value of VARIABLE in the dynamic environment ENVIRONMENT
 * in RESULT.  If ENVIRONMENT is NULL, the login name of the process's
 * uid is used to access the variable.  Upon success, the string returned
 * in RESULT must be freed by the caller;  otherwise, RESULT will be nulled.
 *
 * ZGetenv returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 *   - ZENV_UNKNOWN_VARIABLE if the dynamic environment could be accessed,
 *     but the variable could not be found;
 */
extern int ZGetenv (char *environment, char *variable, char **result);

/*
 * Update VARIABLE in the dynamic environment ENVIRONMENT with VALUE.
 * If ENVIRONMENT is NULL, the login name of the process's uid is used
 * to access the variable.
 *
 * ZSetenv returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 *   - ZENV_UNKNOWN_VARIABLE if the dynamic environment could be accessed,
 *     but the variable could not be found;
 */
extern int ZSetenv (char *environment, char *variable, char *value);

/*
 * Remove the definition of VARIABLE from the dynamic environment
 * ENVIRONMENT.
 *
 * ZUnsetenv returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 *   - ZENV_UNKNOWN_VARIABLE if the dynamic environment could be accessed,
 *     but the variable could be found;
 */
extern int ZUnsetenv (char *environment, char *variable);

/*
 * Return the contents of the dynamic environment ENVIRONMENT as a
 * null-terminated array of strings in RESULT;  each string in RESULT has
 * the form 'variable=value'.  The caller is responsible for freeing
 * the array holding the strings, and the first string only.
 * Upon an error, RESULT is nulled.
 *
 * Zprintenv returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 */
extern int ZPrintenv (char *environment, char **result[]);

/*
 * Shutdown the dynamic environment ENVIRONMENT.
 *
 * Zcloseenv returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 */
extern int ZCloseenv (char *environment);

/*
```

```
 * Subscribe to the dynamic environment ENVIRONMENT.  The caller will
 * receive notifications each time a variable in the environment changes
 * value, but all notification handling must be done by the caller.
 *
 * This routine is useful if you are using dynamic environments as a part
 * of a much larger program that has its own input select loop.  You can
 * use ZEnvGetFD () to get the file descriptor on which notifications will
 * appear so that it can be added to this select loop.  If your program
 * does not have support for input handling, see ZEnvSubscribeLoop(),
 * which will provide it for you.
 *
 * ZEnvSubscribe returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 */
extern int ZEnvSubscribe (char *environment);

/*
 * Subscribe to VARIABLE in the dynamic environment ENVIRONMENT.  The
 * caller will receive notifications each time VARIABLE changes value
 * in ENVIRONMENT, but all notification handling must be done by the
 * caller.
 *
 * This routine is useful if you are using dynamic environments as a part
 * of a much larger program that has its own input select loop.  You can
 * use ZEnvGetFD () to get the file descriptor on which notifications will
 * appear so that it can be added to this select loop.  If your program
 * does not have support for input handling, see ZEnvSubscribeToVarLoop (),
 * which will provide it for you.
 *
 * ZEnvSubscribe returns:
 *   - ZENV_SUCCESS upon success;
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 *   - ZENV_UNKNOWN_VARIABLE if the dynamic environment could be accessed,
 *     but the variable could not be found;
 */
extern int ZEnvSubscribeToVar (char *environment, char *variable);
extern int ZEnvUnsubscribeToVar (char *environment, char *variable);

/*
 * Subscribe to the dynamic environment ENVIRONMENT.  Each time a
 * variable in ENVIRONMENT changes value, CALLBACK is invoked with
 * the user supplied argument ARG, the variable that changed value,
 * and the new value.
 *
 * Note that ZEnvSubscribeLoop normally never returns; the only way control
 * passes back to the caller is through CALLBACK.  If this isn't the
 * semantics you want, see ZEnvSubscribe ().
 *
 * ZEnvSubscribeLoop may return:
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 */
```

```
extern void ZEnvSubscribeLoop (char *environment, ZEnvClosure *closure);

/*
 * Subscribe to VARIABLE in the dynamic environment ENVIRONMENT.  Each
 * time VARIABLE changes value, CALLBACK is invoked with the user supplied
 * argument ARG, the variable, and its new value.
 *
 * Note that ZEnvSubscribeToVarLoop normally never returns; the only way
 * control passes back to the caller is through CALLBACK.  If this isn't the
 * semantics you want, see ZEnvSubscribeToVar ().
 *
 * ZEnvSubscribeLoopToVar may return:
 *   - ZENV_UNKNOWN_ENVIRONMENT if no dynamic environment server could
 *     be found serving the given environment;
 *   - ZENV_UNKNOWN_VARIABLE if the dynamic environment could be accessed,
 *     but the variable could not be found;
 */
extern void ZEnvSubscribeToVarLoop (char *environment, char *variable,
                                    ZEnvClosure *closure);

/*
 * Receive a new published value VAL for the variable VAR in the
 * dynamic environment ENVIRONMENT.  This routine should be used
 * in conjunction with ZEnvSubscribe and ZEnvSubscribeToVar to
 * receive publish events on subscribed dynamic environment values.
 * Note that the caller is responsible for freeing the memory returned
 * in the parameter variables.
 *
 * ZEnvReceivePublishEvent may return:
 *   - True if an event was found
 *   - False if no event was found
 */
extern int ZEnvReceivePublishEvent (char **envPtr, char **varPtr,
                                    char **valPtr);

/*
 * Return the file descriptor used for sending and receiving notifications.
 * This descriptor will be needed when used with ZEnvSubscribe () and
 * ZEnvSubscribeToVar ().
 */
extern int ZEnvGetInputFD (void);

/*
 * Return the error string associated with the given message.
 */
extern char *ZEnvErrorMessage (int);
```

B libmobisaic.a: The Mobisaic Client Library

The Mobisaic client library provides an interface for Web clients to gain access to the Mobisaic extensions to the Web. It exports the following sets of types and routines:

```
/*
 * The mobisaic client library interface.  These routines can be used
 * to augment a WWW client browser with the Mobisaic Web enhancements.
 *
 * See <a href="http://www.cs.washington.edu/homes/voelker/ \
 *              mobisaic/mobisaic.html">mobisaic.html</a>
 * for an introduction to the mobisaic system.
 *
 * See <a href="http://www.cs.washington.edu/homes/voelker/ \
 *              mobisaic/client_mobisaic.html">client_mobisaic.html</a>
 * for a description of the routines found in this file and a description
 * of how the NCSA Mosaic browser was modified to use these routines.
 *
 */

#ifndef SUCCESS
#define SUCCESS 1
#endif

#ifndef FAILURE
#define FAILURE 0
#endif


/*                                              <a name="MS_Initialize">
 * Initialize the mobisaic client library.
 */
extern void MS_Initialize (int argc, char *argv[]);

/*                                              <a name="MS_UpdateWindow">
 * Update the subscriptions for this window using the information
 * parsed from TEXT.
 *
 * Returns SUCCESS if the document was parsed successfully, or
 * FAILURE otherwise.
 *
 * The mobisaic library does use WINDOWDATA itself;  it is a reference
 * to client browser data that is given back to the client when the
 * mobisaic library invokes a client browser callback.
 */
extern int  MS_UpdateWindow (void *windowData, char *text);

/*                                              <a name="MS_ClearWindow">
 * Clear any subscriptions maintained for this window.  Returns
 * SUCCESS if the subscriptions were successfully cleared, or
 * FAILURE otherwise.
 */
extern int  MS_ClearWindow (void *windowData);

/*                                              <a name="MS_IsADynamicURL">
 * Return non-zero if the given URL is a syntactically correct
 * dynamic URL, i.e., it has at least one dynamic variable reference
 * and those references have the proper syntax.  Otherwise, return 0.
 */
extern int  MS_IsADynamicURL (char *url);

/*                                              <a name="MS_ResolveEnvVars">
 * Convert DYNAMIC_URL into a normal URL by resolving any variable
 * references it may have.  If successful, the normal URL is returned
 * through OUT_URL, and the caller is responsible for freeing the memory
 * associated with it.
 *
 * Returns SUCCESS if all variable references were resolved successfully,
 * or FAILURE otherwise.
 */
extern int  MS_ResolveEnvVars (char *dynamicURL, char **outURL);

/*                                              <a name="MS_Callback">
 * Callback procedures that the mobisaic library uses to invoke
 * a defined set of operations in the client browser.  The procedures
 * must accept two parameters:
 *
 * WINDOWDATA:  A reference to client browser data passed
 *              into MS_UpdateWindow.
 * URL:         A reference to a URL to be used in the callback, or NULL
 *              if a URL doesn't apply.
 */
typedef void (*MS_Callback) (void *windowData, char *url);

/*                                              <a name="MS_SetCloseCallback">
 * A browser routine that, given a window reference, closes that window.
 */
extern void MS_SetCloseCallback (MS_Callback);

/*                                              <a name="MS_SetGotoCallback">
 * A browser routine that, given a window reference and a URL, loads
 * the URL in that window.
 */
extern void MS_SetGotoCallback (MS_Callback);

/*                                              <a name="MS_SetReloadCallback">
 * A browser routine that, given a window reference, reloads the document
 * in that window.
 */
extern void MS_SetReloadCallback (MS_Callback);

/*                                              <a name="MS_SetSpawnCallback">
 * A browser routine that, given a window reference and a URL, spawns
 * a new window and loads the URL in it.
 */
extern void MS_SetSpawnCallback (MS_Callback);

/*                                              <a name="MS_GetWindowCB">
 * A callback procedure that the mobisaic library uses to get
 * a reference to the "current window" in the client browser.
 */
```

```
typedef void * (*MS_GetWindowCB) (void);

/*                                      <a name="MS_SetGetCurrentWinCallback">
 * Set the callback procedure that the mobisaic library can use
 * to obtain a reference to the "current window" in the client
 * browser.  The reference returned must be a valid first argument
 * to the callbacks above.
 */
extern void MS_SetGetCurrentWinCallback (MS_GetWindowCB);

/*                                              <a name="MS_GetInputFD">
 * The input descriptor on which the mobisaic library receives
 * notifications.  This routine is usually used to access the
 * descriptor for subsequent placement in the client's input select
 * loop.
 */
#define MS_GetInputFD() ZEnvGetInputFD ()

/*                                              <a name="MS_InputCallback">
 * The mobisaic input callback that should be invoked when
 * a client browser detects input on the mobisaic input
 * descriptor.
 */
extern void MS_InputCallback (void);
```